<Seno>

<Senozza>
Software Architecture Document

Version <2.0>

Revision History

Date	Version	Description	Author
<10/12/2022>	<1.0>	This document describes the process to construct the architecture of our school moodle web. Consisting of a database diagram, a package diagram, and a logical view diagram.	Seno Team
<23/12/2022>	2.0	In this report, we have many updates in the use-case diagram, enhancements in the logical view parts and we fully completed two parts: Deployment and Implementation View.	Seno Team

Table of Contents

Software Architecture Document

Introduction

In this document, we will describe the full architecture design of our School Moodle website. It consists of the use case model which is the diagram to describe the interaction between three users (Student, Administrator and Lecturer) and our system.

In the next section (Architecture Goals and Constraints), we will show some description about important information of our project such as: Scope, Goals, Constraints, Team Strucute, used tools and some strategies that we use to develop this project.

Use-cases model will represent all main use-cases which are important in our program.

Final Section (Logical View) will show all detailed components of the software.

Architectural Goals and Constraints

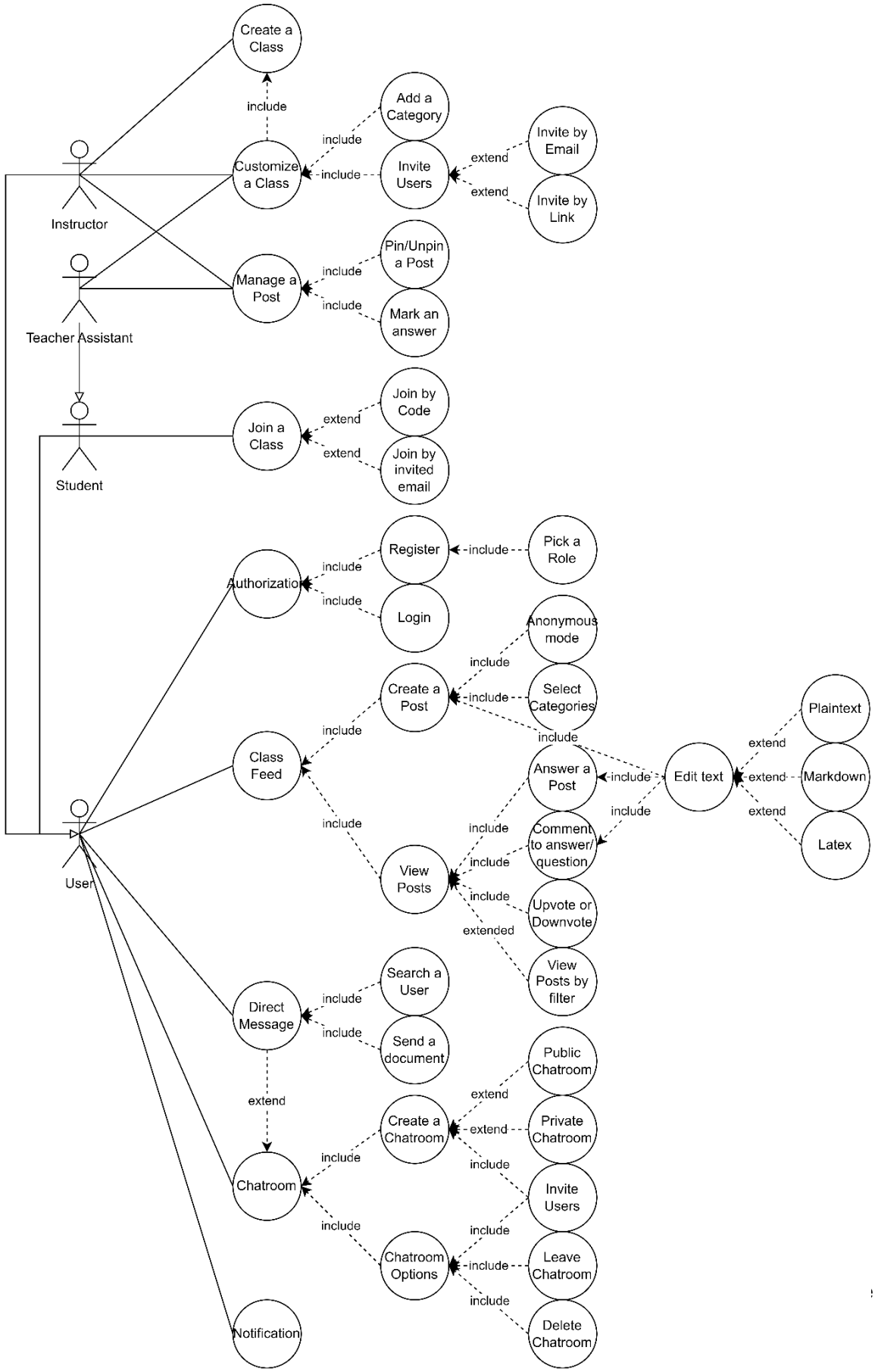
Main constraints:

- Safety: Our website has a very useful interface for the users, attractive UI design and very easy to use. When they access our website, they can smoothly use some basic functions such as: clicking, scrolling and typing.
- Security: We will prevent attacks from all levels, make sure that our database cannot be changed by attacking outside. We also have a backup server to recover all the data when our web is collapsed. We also have some prevention for basic web attacks: DDOS, SQL injection, Cross-site scripting,..
- Privacy: This is the most important. We will make sure that the privacy data of all users are not leaked and the users from our system cannot see any private information from other users.
- Design and implementation strategy: We will follow the RUP and SCRUM strategy to assign work for all members, and try to maximize the performance of our program by helping the other teammates when they have some troubles.
- Statibility: Our server must be efficient enough, especially in some worst case where there are lots of users accessing our website, all requests must be responded within one minute. When some task is completed, there must be some notifications to announce the user.

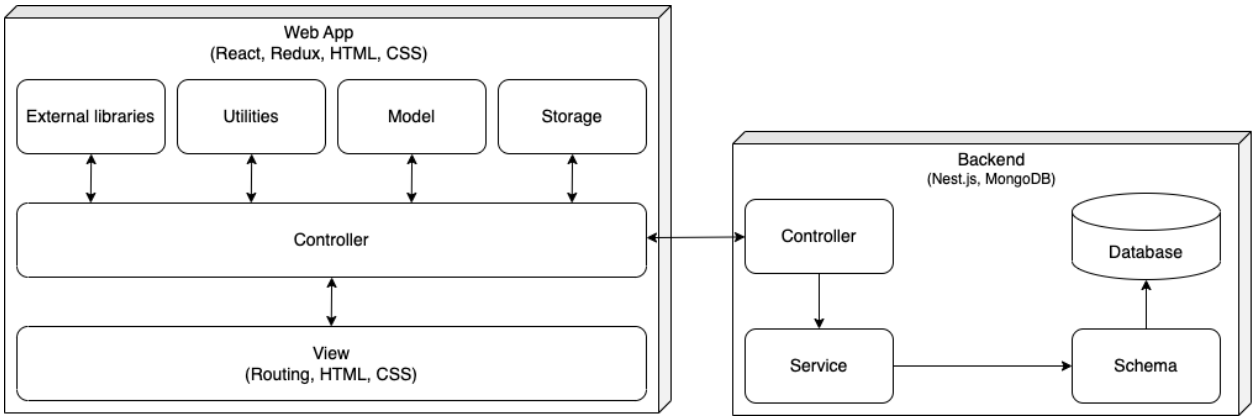
Development tools:

- We will use web draw.io to design the component diagram
- For designing the UI of our web, we will use Figma
- For designing the database diagram, we will use Microsoft SQL server 2019 to construct the database diagram.
- We use visual.paradigm for designing the package diagram following the MVC architecture.
- We use Typescript, React, Fastify, Mongodb, Nest.js for our development.

Use-Case Model


Create a Class
include
Add a Category
include
Invite by Email
extend
Customize a Class
include
Invite Users
extend
Invite by Link
Instructor
Pin/Unpin a Post
include
Manage a Post
include
Mark an answer
Teacher Assistant
Join by Code
extend
Join a Class
extend
Join by invited email
Student
Register
include
Pick a Role
include
Authorization
include
Login
Anonymous mode
include
Create a Post
include
Select Categories
include
Plaintext
extend
include
Class Feed
include
Answer a Post
include
Edit text
extend
Markdown
include
include
Comment to answer/ question
extend
Latex
include
View Posts
include
Upvote or Downvote
extended
View Posts by filter
User
Search a User
include
Direct Message
include
Send a document
Public Chatroom
extend
extend
Create a Chatroom
extend
Private Chatroom
extend
include
Chatroom
include
Invite Users
include
include
Chatroom Options
include
Leave Chatroom
include
Delete Chatroom
Notification

Logical View



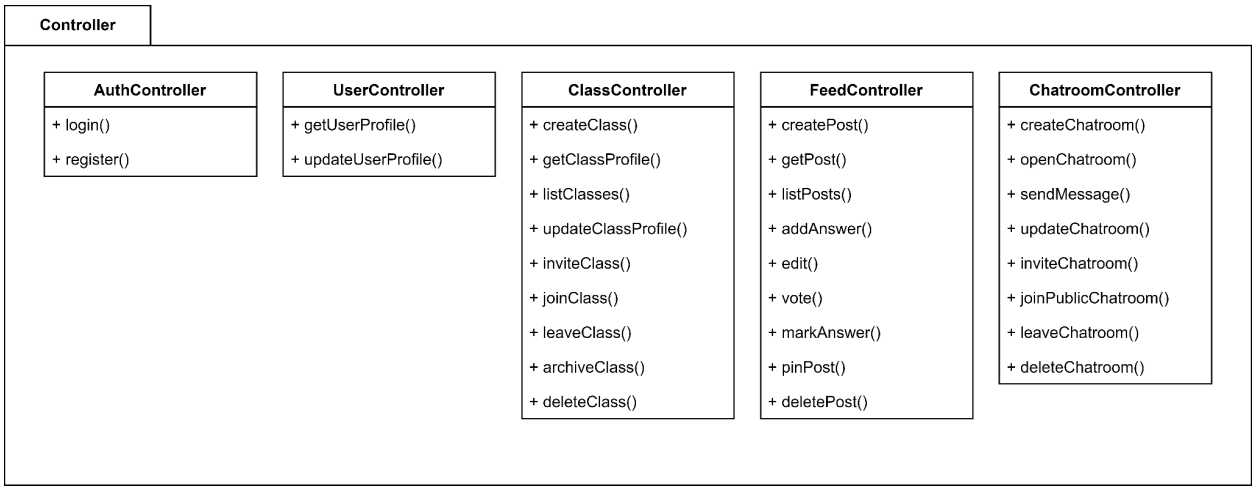
Component: Web app

- Framework: React
- Programming language: Typescript
- Provide a user interface to interact with the server.

Component: Backend

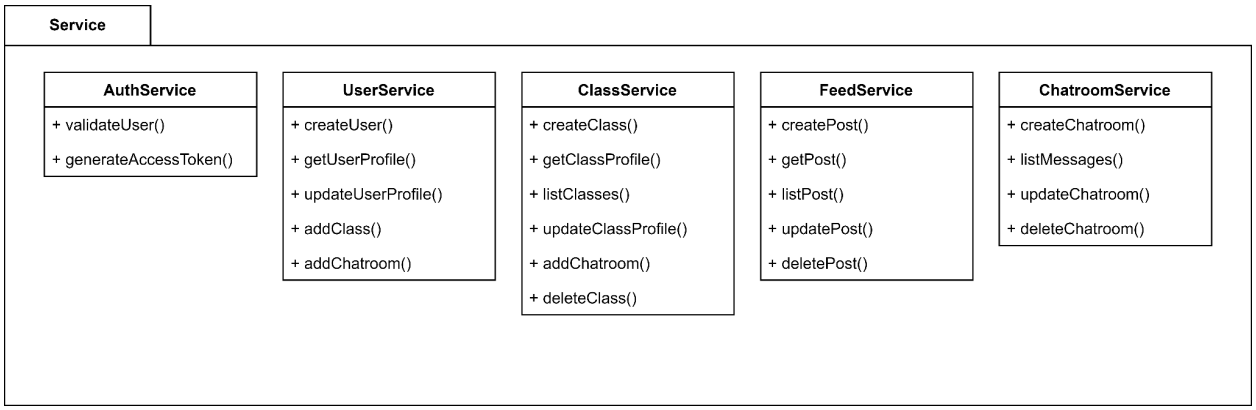
- Framework: NestJS
- Programming language: Typescript
- Handling requests from Web App and querying data from Database.

Component: Backend::Controller



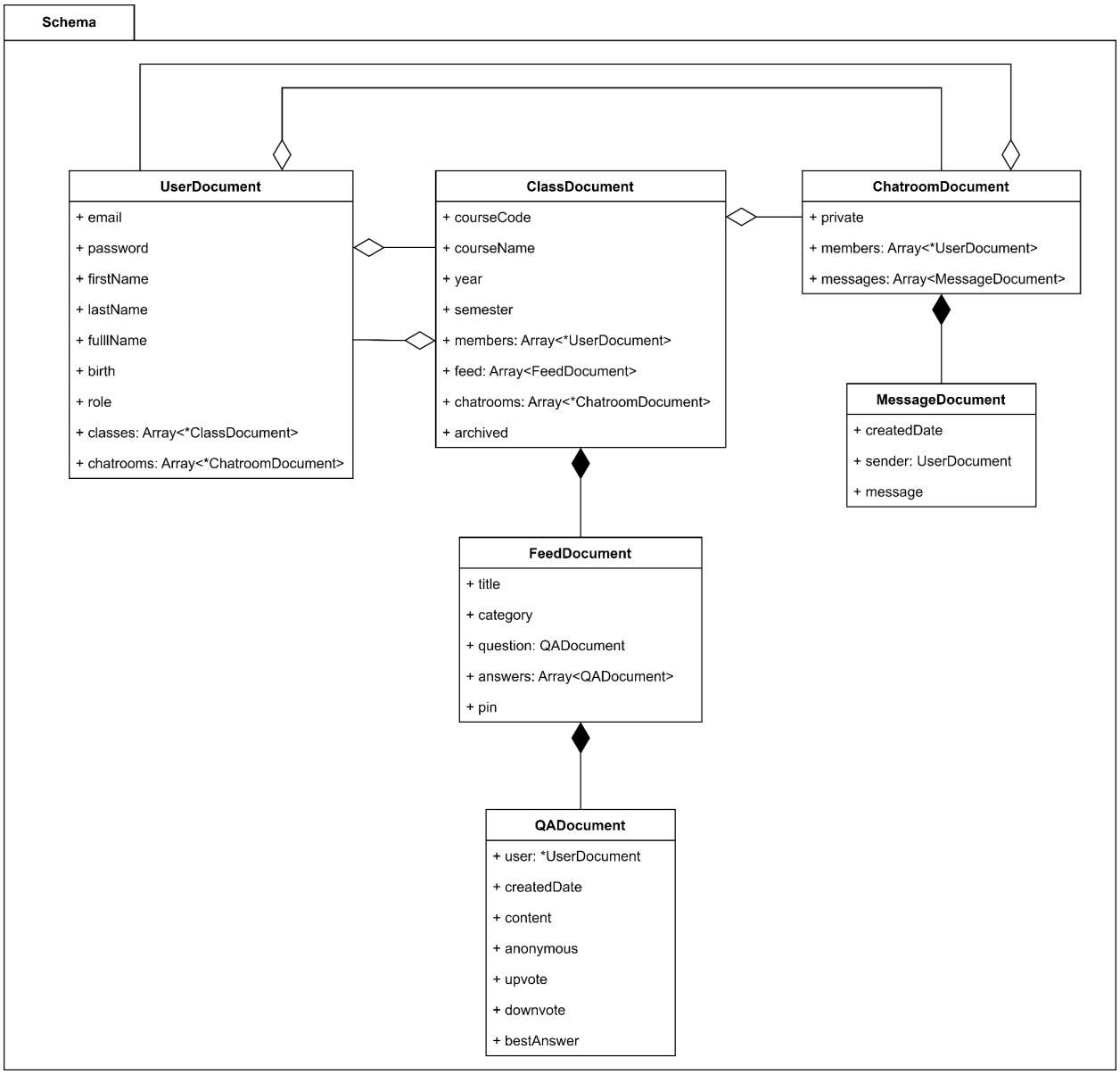
- **AuthController:** Handle authentication requests from clients
- **UserController:** Handle requests for retrieving or updating data from Users collection
- **ClassController:** Handle requests for retrieving or updating data from Classes collection
- **FeedController:** Handle requests for retrieving or updating data from Feeds collection
- **ChatroomController:** Handle requests for retrieving or updating data from Chatrooms collection

Component: Backend::Service



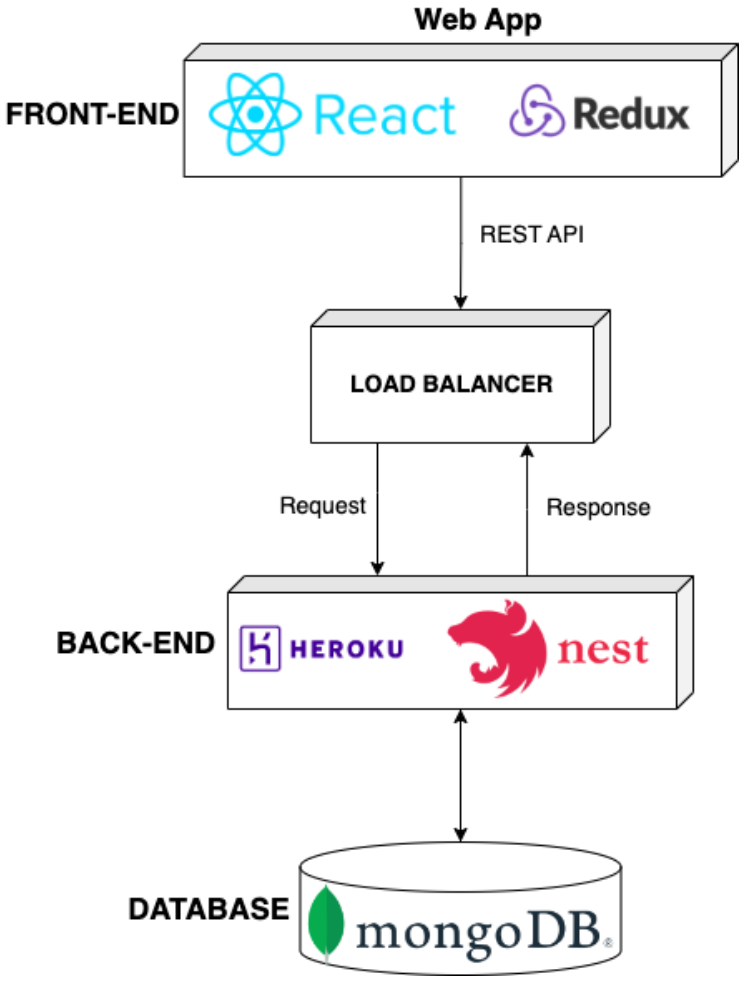
- **AuthService:** Provide function for authentication
- **UserService:** Provide function for user query
- **ClassService:** Provide function for class query
- **FeedService:** Provide function for feed query
- **ChatroomService:** Provide function for chatroom query

Component: Backend::Schema



- Using MongoDB
- There are 3 main document: *User, Class* and *Chatroom*
    - *QA* is subdocument of *Feed* and *Feed* is subdocument of *Class*
    - *Message* is subdocument of *Chatroom*
- A *User* may attend in many *Classes* and *Chatrooms*
- A *Class* can have many *Users*(members) and *Chatrooms* inside of it, each *Feed* is contained in a particular *Class*.
- A *Chatroom* may have many *Users*(members), and each *Message* is in a particular *Chatroom*

Deployment



Implementation View

Frontend:

```
src/app
├── App.css
├── App.test.tsx
├── App.tsx
├── app/
│   ├── hooks.ts
│   └── store.ts
├── components/
│   ├── chatBox.tsx
│   ├── comment.tsx
│   ├── markdownEditor.tsx
│   ├── markdownPreview.tsx
│   └── navBar.tsx
├── constants/
│   └── index.ts
```

```
├── features/
│   ├── auth/
│   │   ├── recoverPassword.tsx
│   │   ├── signIn.tsx
│   │   ├── signUp.tsx
│   │   └── slice.ts
│   ├── chatroom/
│   │   ├── chatRoom.tsx
│   │   └── slice.ts
│   ├── class/
│   │   ├── classFeed.tsx
│   │   ├── classSetting.tsx
│   │   ├── createClass.tsx
│   │   └── slice.ts
│   └── notification/
│       ├── Notifications.tsx
│       └── slice.ts
├── index.css
├── index.tsx
├── logo.svg
├── pages/
│   ├── Dashboard.tsx
│   ├── recoverPassword.tsx
│   ├── signIn.tsx
│   └── signUp.tsx
├── react-app-env.d.ts
├── reportWebVitals.ts
├── setupTests.ts
└── utils/
    └── index.ts
```

Backend:

```
src/server
│   app.module.ts
│   main.ts
│
├───auth
│   │   auth.controller.ts
│   │   auth.module.ts
│   │   auth.service.ts
│   │
│   ├───guards
│   │       jwt-auth.guard.ts
│   │
│   └───strategies
│           jwt.strategy.ts
│
├───chatroom
│   │   chatroom.controller.ts
│   │   chatroom.module.ts
│   │   chatroom.service.ts
```

```
│   │
│   └──message
│          message.controller.ts
│          message.module.ts
│          message.service.ts
│
├──class
│   │   class.controller.ts
│   │   class.module.ts
│   │   class.service.ts
│   │
│   └──feed
│       │   feed.controller.ts
│       │   feed.module.ts
│       │   feed.service.ts
│       │
│       └──post
│              post.controller.ts
│              post.module.ts
│              post.service.ts
│
├──common
├──schemas
│      chatroom.schema.ts
│      class.schema.ts
│      user.schema.ts
│
└──user
       user.controller.ts
       user.module.ts
       user.service.ts
```